Generator 抽象实现报告

2026 春季人工智能大赛 · InfiniTrain

参赛者	陈海源
参赛小组	Anonymous
GitHub ID	surprisely
竞赛分支	2026-spring-surprisely-T2-2-1
最终提交	136b534036b7
完成日期	2026-07-12

最终门禁

CPU Generator 41/41 · CUDA Generator 16/16 · 双 GPU 10/10 · ASan+UBSan 41/41

实现范围

本次实现补齐了框架内 Generator 基础设施，并提供了 CPU 与 CUDA 后端随机入口。CUDA 张量随机填充已接入 kernel-side seed/offset 生成路径。

已完成：

- Generator 用户侧句柄。
- GeneratorImpl 多态基类。
- CPUGeneratorImpl。
- CUDAGeneratorImpl，包含 per-device seed/offset 状态。
- CPU 默认 Generator。
- CUDA 按设备维护默认 Generator。
- ManualSeed / ManualSeedAll 统一 seed 入口。
- Seed / InitialSeed。
- GetState / SetState 状态保存与恢复。
- Tensor::Uniform 接入显式/默认 Generator。
- nn::init::{Uniform, Normal, KaimingUniform} 接入 Generator。
- nn::function::{Rand, Randn} 作为随机张量生成入口。
- 显式 Generator 与 tensor 后端类型一致性校验；同属 CUDA 后端时允许跨 device index 使用。
- CPU/CUDA 公共 Uniform / Normal 当前均仅支持 float32；CUDA 随机填充 kernel 使用 Philox-style seed/offset 生成。
- CPU 和 CUDA 相关测试。
- 跨进程复现检查脚本 scripts/check_generator_reproducibility.sh。

外部参考与 AI 工具披露

本项目参考 PyTorch 的 Generator/GeneratorImpl 分层、默认/显式 Generator、manual seed、state 保存恢复及 CUDA seed/offset 组织原则，但代码按 InfiniTrain 现有 Tensor、Device、内存和 stream API 独立实现，未复制 PyTorch 实现代码，也不承诺逐 bit 一致。

本项目由参赛者主导并实际参与，使用 OpenAI Codex 作为辅助开发工具。CPU/CUDA 构建、测试、双物理 GPU、跨进程 digest 与 sanitizer 证据均来自目标环境真实执行。具体说明见随提交材料提供的 REFERENCE.md。

赛题要求与验收证据

赛题要求	实现位置	验证证据
统一 Generator 抽象	include/generator.h、src/generator.cc	CPU/CUDA 统一接口测试
CPU / CUDA GeneratorImpl	CPUGeneratorImpl、CUDAGeneratorImpl	CPU/CUDA 定向测试与从零构建
seed 与 state 管理	ManualSeed、Seed、InitialSeed、GetState、SetState	同/异 seed、状态重放、损坏 state 测试
默认 Generator	CPU singleton、CUDA per-device map	默认分发、共享状态、双 GPU 隔离测试
统一全局 seed	ManualSeed / ManualSeedAll	现有及懒创建 CUDA Generator 重置测试
显式 Generator	Rand/Randn/初始化接口可传 Generator	显式 Generator 不推进默认状态测试
初始化类随机算子	Uniform、Normal、KaimingUniform	CPU/CUDA 初始化复现测试
框架随机生成调用	Rand、Randn	CPU/CUDA 可复现、推进与恢复测试
线程安全	Generator 内部 mutex	显式及默认 Generator 并发 offset 测试

赛题要求	实现位置	验证证据
跨设备接口统一	GetDefaultGenerator(device)	两张物理 GPU 的状态隔离测试
报告与可复现性	本报告、GTest 与复现脚本	定向测试与跨进程 digest 比较

设计说明

Generator 只持有 std::shared_ptr<GeneratorImpl>，对外暴露统一接口，不暴露后端 RNG engine。CPU 和 CUDA 后端分别由 CPUGeneratorImpl、CUDAGeneratorImpl 实现，具体实现类定义放在源文件内。公共头文件保留抽象接口、工厂函数，以及 CUDA random kernel 使用的 offset reservation 入口；不暴露 std::mt19937、curand 或 Philox 状态对象。

ReserveRandomOffset 当前是 CUDA kernel 接入所需的 offset 预留语义，只对 CUDA Generator 有效； CPU Generator 不维护 offset，调用该接口会明确失败。CPU 随机消费通过受 mutex 保护的 FillUniform / FillNormal 推进 engine state。该接口是当前实现的后端接入边界，不代表 CPU 与 CUDA 必须采用相同的内部状态组织方式。

ManualSeed(uint64_t) 会用 seed 的高/低 32 位共同初始化 CPU engine，避免 64 位 seed 高位被截断。CPU 后端使用 std::mt19937 管理随机序列。CUDA 后端只维护 seed 与 offset；CUDA tensor 上的 Uniform / Normal 随机填充直接调用 CUDA kernel，根据 Generator 预留的 (seed, offset) 生成随机数，避免不同调用复用同一随机段。

CUDA Generator 的 host FillUniform / FillNormal 与 CUDA kernel 使用相同的 Philox counter 映射和 offset 消费规则，不再维护独立的 host mt19937 fallback engine。这样无论先通过 host 接口消费、再进入 kernel，还是完全使用 kernel，后续随机区间都由同一个 offset 描述且不会重叠。Normal 的 host/device 浮点数学实现可能产生末位差异，因此只承诺随机流区间和状态语义一致，不承诺 host/device 逐 bit 一致。

默认 Generator 首次创建时使用进程级非固定 seed；ManualSeedAll(seed) 会重置 CPU 默认 Generator、所有已创建 CUDA 默认 Generator，并记录默认 seed，使后续懒创建的 CUDA 设备 Generator 也使用同一个 seed。GetDefaultGenerator(device) 会根据设备类型分发到 CPU 默认 Generator 或对应 CUDA 设备默认 Generator；ManualSeed(seed) 作为 ManualSeedAll(seed) 的别名提供统一 seed 入口。

State 设计

GetState 返回 std::vector<uint8_t>，内部包含：

- 后端类型标识。
- 初始 seed。
- CUDA state 来源 device index（仅 CUDA Generator，用于格式和来源范围校验，不限制恢复目标 index）。
- CUDA offset（仅 CUDA Generator，记录 kernel-side 随机流已消费的位置）。
- CPU std::mt19937 序列化状态。

SetState 会校验后端类型并严格解析完整格式。seed、CUDA offset 和其他整数 token 不接受负值、溢出或夹杂字符；任何尾随内容（包括纯空白）都会拒绝。CPU engine 必须恰好包含 624 个不超过 uint32_t 的 state word 和 1 个不超过 624 的 position token，且 state word 不能全零。CUDA state 的来源 device index 必须位于 int8_t 非负范围，但同属 CUDA 后端的 Generator 可跨 index 恢复 seed/offset，与 PyTorch 的同后端 state 迁移语义一致。

测试覆盖了损坏、空、截断、尾随垃圾/纯空白、负无符号字段、整数夹杂字符、CPU engine token 数量/word 范围/position/all-zero、CPU/CUDA 后端不匹配、CUDA 非法来源 index、CUDA 跨 index 重放和 offset 损坏等路径。

state header 带有明确版本。CUDA state 在移除 fallback engine 后升级为 v3，旧版本会被明确拒绝，而不是被错误解析。CPU engine 的文本序列化依赖 C++ 标准库实现，适合相同构建环境内保存与恢复；本实现不承诺跨标准库、跨架构长期存档兼容。CUDA state 只包含整数语义字段，可移植性更强，但仍通过版本 header 管理格式演进。

算子接入

初始化类随机算子：

- nn::init::Uniform
- nn::init::Normal
- nn::init::KaimingUniform
- Tensor::Uniform

随机张量生成入口：

- nn::function::Rand
- nn::function::Randn

未显式传入 Generator 时，接口会根据 tensor/device 自动使用当前设备默认 Generator；显式传入时使用用户指定 Generator，不推进默认 Generator 状态。

当前 CPU/CUDA 随机填充公共入口均只接受 float32 tensor，非 float32 会在写入前显式拒绝。Uniform 会检查上下界有限、顺序有效且区间宽度不超过 float 最大值；相等上下界返回常数但仍推进随机状态。Normal 支持 std == 0，结果恒为 mean，同时保持与普通 Normal 一致的 state/offset 推进语义。

空的 Uniform/Normal（包括 Rand/Randn）tensor 调用仍执行分布参数和 Generator backend 校验，然后在分配随机区间或复制数据前返回，因此既不推进状态，也不会触发 0 字节 null-pointer memcpy。CUDA kernel 的元素索引使用 64-bit 中间计算。

源码兼容性与迁移

Tensor::Uniform 和 nn::init::{Uniform, Normal, KaimingUniform} 的显式随机源参数由 std::optional<std::mt19937> 更新为 std::shared_ptr<Generator>。省略该参数的调用不需要修改； 原先显式传入 std::mt19937 的调用方需改为创建 MakeCPUGenerator(seed)，或在 CUDA tensor 场景中 创建 MakeCUDAGenerator(device_index, seed)，再传入对应接口。旧 mt19937 engine state 不能直接导入 新 Generator；迁移后应使用 GetState / SetState 保存和恢复新随机流状态。

仓库中的 example/gpt2/checkpoint_loader.cc 与 example/llama3/checkpoint_loader.cc 仍保留各自局部 std::mt19937。它们属于示例 checkpoint loader 的独立随机路径，不经过本次改造的框架初始化与 Rand/Randn 核心调用链；本实现不声称已经统一仓库内所有 RNG 使用点。

Dropout 范围说明

赛题将 Dropout、Rand、Randn 并列为训练期随机场景示例，最终验收要求是至少接入一类框架随机生成调用。当前框架没有既有 Dropout 算子，本实现选择 Rand/Randn 完成 Generator 链路验证。为避免将不完整的 training/eval、mask scaling、autograd 或 CUDA 语义带入主线，本次不额外创建半成品 Dropout；后续完整 Dropout 可直接复用默认/显式 Generator 解析和 CUDA offset reservation 机制。

测试结果

远端环境：

- 路径：/path/to/InfiniTrain
- CMake：3.28.4
- GCC/G++：13.4.0
- CUDA：12.4
- GPU：2 × NVIDIA GeForce RTX 2080 Ti（每卡 11264 MiB）

- 最终验证日期：2026-07-12

验证命令：

```
# CPU-only build
cmake -S . -B build-cpu -G Ninja -DUSE_CUDA=OFF -DUSE_NCCL=OFF -DBUILD_TEST=ON \
-DCMAKE_C_COMPILER=gcc \
-DCMAKE_CXX_COMPILER=g++
jobs=$(( $(awk '/MemAvailable/ {print $2}' /proc/meminfo) / 512000 ))
nproc=$(nproc)
[ "$jobs" -gt "$nproc" ] && jobs=$nproc
[ "$jobs" -lt 1 ] && jobs=1
cmake --build build-cpu -j"$jobs"
cd build-cpu && ctest -L cpu -j"$jobs" --output-on-failure

# CUDA build
cd /path/to/InfiniTrain
cmake -S . -B build-cuda -G Ninja -DUSE_CUDA=ON -DUSE_NCCL=OFF -DBUILD_TEST=ON \
-DCMAKE_C_COMPILER=gcc \
-DCMAKE_CXX_COMPILER=g++ \
-DCMAKE_CUDA_COMPILER=/usr/local/cuda/bin/nvcc \
-DCMAKE_CUDA_HOST_COMPILER=g++
cmake --build build-cuda -j"$jobs"
cd build-cuda && ctest -L cuda -j"$jobs" --output-on-failure
```

最终结果（2026-07-12）：

测试范围	总数	通过	跳过	失败
CPUGeneratorTest.*	41	41	0	0
CPU 全量标签测试	242	231	11	0
CUDAGeneratorTensorTest.*（双卡）	16	16	0	0
CUDA 标签测试	9	9	0	0

上述数字来自定稿代码的最终测试日志。CPU 标签的 11 个 skipped 项目是既有资源/环境相关用例，不是 Generator 测试失败。跨物理设备用例以 --gtest_repeat=10 --gtest_break_on_failure 连续运行 10 次，10/10 passed。CPU 与 CUDA 均完成定向测试；跨进程 Kaiming 初始化、Rand、Randn digest 两次均为 ec0f626ebd167804。

另以 AddressSanitizer + UndefinedBehaviorSanitizer 运行 CPU Generator 定向集，41/41 passed 且未报告 sanitizer 错误。原始日志随测试证据留存；SHA-256：a43749c56429c555e27af3255c309f60a1324315f71c8c6d9b0eb961b0d23707。

定向复现：

```
cd /path/to/InfiniTrain
build-cpu/tests/tensor/test_tensor_cpu_only --gtest_filter='CPUGeneratorTest.*'
build-cuda/tests/tensor/test_tensor_cuda_only --gtest_filter='CUDAGeneratorTensorTest.*'
scripts/check_generator_reproducibility.sh build-cpu/tests/tensor/test_tensor_cpu_only
```

跨进程复现检查会分别启动两次测试进程，对固定 seed 下 Kaiming 初始化、Rand、Randn 的结果计算稳定 digest 并比较：

```
scripts/check_generator_reproducibility.sh build-cpu/tests/tensor/test_tensor_cpu_only
```

参考设计与接口映射

主要参考：

- [PyTorch torch.Generator 文档](https://docs.pytorch.org/docs/stable/generated/torch.Generator.html)：公共接口语义。

- [PyTorch ATen/core/Generator.h](https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/core/Generator.h)：用户句柄与实现分层。
- [PyTorch CPUGeneratorImpl.cpp](https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/CPUGeneratorImpl.cpp)：CPU seed/state 管理。
- [PyTorch CUDAGeneratorImpl.cpp](https://github.com/pytorch/pytorch/blob/main/aten/src/ATen/cuda/CUDAGeneratorImpl.cpp)：逐设备默认 Generator、Philox seed/offset 思路。
- [PyTorch randomness note](https://docs.pytorch.org/docs/stable/notes/randomness.html)：可复现性的范围与限制。

InfiniTrain	PyTorch 对应语义	对齐范围
Generator / GeneratorImpl	at::Generator / backend impl	句柄与多态实现分层
ManualSeed	manual_seed	设置初始 seed 并重置序列
Seed	seed	生成随机 seed 并重置序列
InitialSeed	initial_seed	返回初始化 seed
GetState / SetState	get_state / set_state	保存与恢复序列位置
GetDefaultGenerator	default CPU/CUDA generators	未显式传入时按设备选择
ManualSeedAll	全局 seed 控制语义	重置 CPU、已有及未来 CUDA 默认 Generator
ReserveRandomOffset	CUDA Philox offset reservation	为每次 kernel 分配不重叠随机区间

本项目参考接口语义和状态组织原则，没有复制 PyTorch 的实现代码，也不要求底层算法或输出逐 bit 一致。

与 PyTorch 语义对齐

已对齐：

- 用户可持有显式 Generator。
- 未传 Generator 时使用当前设备默认 Generator。
- 全局 seed 入口控制默认 Generator。
- 未显式设 seed 时使用进程级非固定默认 seed；调用 ManualSeed 后行为可控且可复现。
- 同 seed、同调用顺序结果可复现。
- 多次调用会推进 Generator 状态。
- GetState / SetState 可恢复随机序列位置。
- CPU 默认 Generator 与 CUDA 逐设备默认 Generator 分离。
- CUDA state 显式记录 offset，CUDA tensor 随机 kernel 消费该 offset 分配的随机流区间。
- offset reservation 在多线程并发下保持原子、连续且互不重叠。
- CUDA 随机流在拆分为多次 kernel 调用后与单次连续调用一致，并通过 Uniform 范围及 Normal 均值/方差的统计 smoke test。
- CUDA host 消费与 kernel 消费共享同一个 seed/offset 随机流描述。
- 固定 seed 下的初始化、Rand、Randn 在两个独立进程中产生一致 digest。
- Normal 的 std == 0 返回 mean，同时继续推进 Generator state/offset。
- 同属 CUDA 后端的显式 Generator 可跨 device index 用于 tensor，CUDA state 也可跨 index 恢复；逐设备默认 Generator 仍相互独立。
- Uniform 的有限边界、区间宽度和相等上下界语义与 PyTorch 基本一致。

未对齐或后续工作：

- 当前 CUDA kernel 使用项目内实现的 Philox-style 生成逻辑，不承诺与 PyTorch 逐 bit 一致。

- CPU/CUDA 公共随机填充目前均只覆盖 float32 Uniform / Normal 路径；其他 dtype 后续可扩展，当前会显式拒绝。
- CUDA state 记录 seed/offset，但尚未实现完整 PyTorch 风格的 graph-safe RNG state、capture tracking 等高级语义。
- 已在两张物理 GPU 上验证 CUDA 0/1 默认 Generator 独立推进；一个设备上的真实随机 kernel 调用不会改变另一设备的 Generator state。
- ReserveRandomOffset 是当前 CUDA 后端接入语义，CPU Generator 不支持该调用。
- 显式 std::mt19937 初始化参数已迁移为 std::shared_ptr<Generator>，外部旧式调用需要源码适配。
- GPT2/LLaMA3 示例 checkpoint loader 的局部 mt19937 不在本次核心随机调用链改造范围内。